ARMY RESEARCH LABORATORY

A Programmer's Guide to the Overwatching Fires Behavior

by MaryAnne Fields, MyVan Hoang Baranoski, and B. Tom Haug

ARL-TR-3548 **July 2005**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-3548 **July 2005**

A Programmer's Guide to the Overwatching Fires Behavior

MaryAnne Fields, MyVan Hoang Baranoski, and B. Tom Haug
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
July 2005	Final	October 2003–October 2004

4. TITLE AND SUBTITLE
A Programmer's Guide to the Overwatching Fires Behavior

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
MaryAnne Fields, MyVan Hoang Baranoski, and B. Tom Haug

5d. PROJECT NUMBER
622618AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRD-ARL-WM-BF
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-3548

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes the software modules required to demonstrate an overwatching fires (OWF) behavior on a team of laboratory robotic platforms. The modules are divided into three types—servers, an operator control unit (OCU), and the OWF application. Servers are independent software programs that communicate with sensors and actuators on-board the robot. The OCU is an independent process that allows operators to start, modify, and stop the OWF behavior. The OWF application consists of several modules that control sensing, communication, movement, and shooting for each of the robots in the team.

15. SUBJECT TERMS
robot, behavior algorithms, overwatching fires

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			MaryAnne Fields
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	38	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6675

Contents

List of Figures

List of Tables

1. Introduction

One of the goals of the U.S. Army ground robotics research program is to develop individual and group behaviors that allow the robot to contribute to battlefield missions such as reconnaissance. As a part of this research program, at the Weapons Technology Analysis Branch of the U.S. Army Research Laboratory (ARL), we have developed a behavior to demonstrate aspects of an overwatching fires, hereafter referred to as OWF, mission. The behavior is a cooperative mission – a team of ground robots (the current behavior is limited to the use of two ground robots, but could be expanded to include the use of air assets, as well as additional ground robots) and human operators work together to protect an area from enemy incursion. Human operators have a limited role in this behavior. They designate the area of interest for the robot and may need to confirm targets before the robots fire upon them. There are two distinct roles for robot team members – observers and shooters. Roles are assigned to robots before the mission starts; robots cannot switch roles after the mission begins. The observers watch for enemy units in the designated area. Once enemy units have been identified, the shooters move into position and fire upon enemy units detected by the observers. After the shooter has fired on its target, it may move to another firing position to await its next target. The mission continues until the enemy unit leaves the area, sufficient damage has been inflicted, or the OWF unit receives a new mission.

Initially, we used the battlefield simulation tool One Semi-Automated Forces Test Bed (OTB) to develop an OWF algorithm that was not tied to a specific robotic hardware configuration. This work is documented in Fields.[1] The primary focus of this report is the robotic implementation of the OWF algorithm, although we discuss aspects of the OTB implementation as well. In the remainder of this section, we provide a detailed description of the algorithm and a discussion of the robotic hardware used in this work. Sections 2 and 3 are programmer's guide describing the software developed to implement the OWF behavior algorithm on a specific type of robotic platform produced by iRobot. Documenting the OWF algorithm provides a detailed example for other researchers trying to develop robotic algorithms. The last section is a discussion of planned experiments for the OWF maneuver.

1.1 The OWF Algorithm

A process flowchart for the OWF behavior is shown in figure 1. There are three major tasks in the figure: planning, observing, and shooting. Generally, for the robotic implementation, each of these major tasks is handled by independent computer programs which must communicate to

[1] Fields, M. Developing an Overwatching Fires Mission for a Team of Unmanned Ground Vehicles. In *Performance Metrics for Intelligent Systems, '03*; NIST Special Publication 1014; National Institute of Standards and Technology: Gaithersburg, MD, 2003.

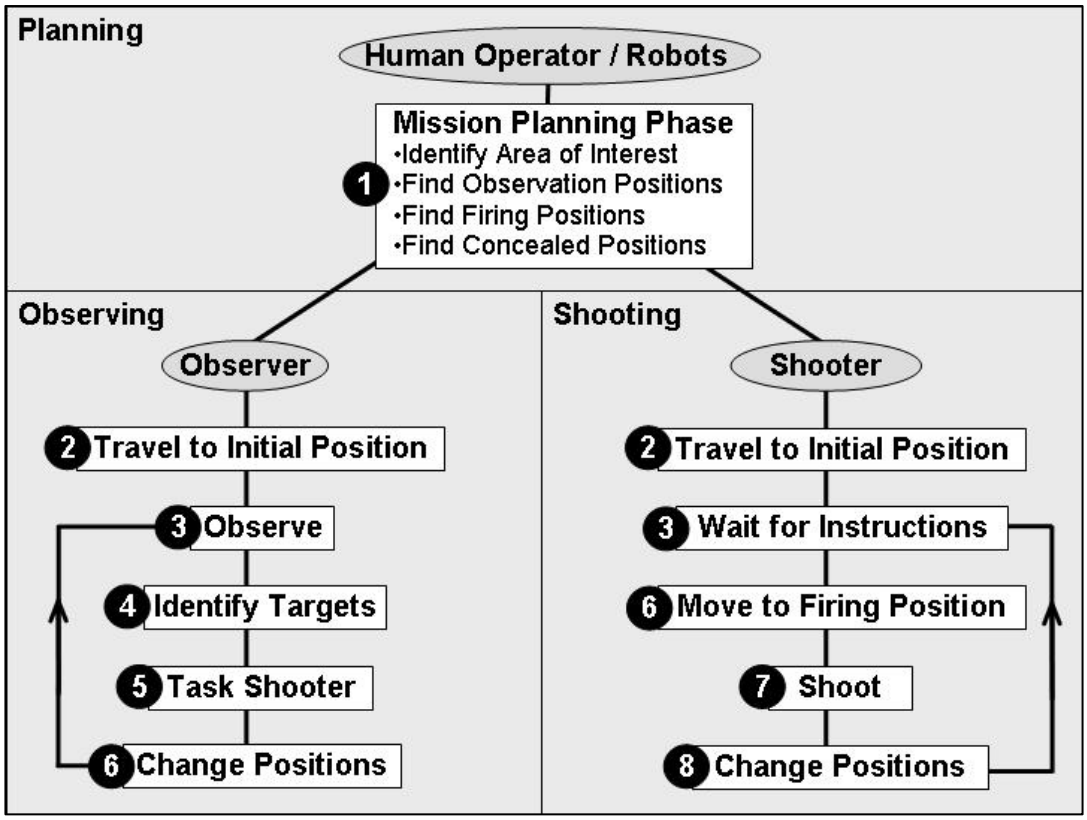

Figure 1. The OWF process flowchart.

accomplish the overall mission objective. The ovals in the diagram indicate which team members execute the major task. The numbers on the left side of the subtask boxes in figure 1 indicate the required processing order. Numbers appearing more than once indicate subtasks that can occur simultaneously. The major task of the behavior is team-level mission planning. In the OTB simulation, the human operator designates the area of interest, usually a road segment that the team is to protect. The simulated robots perform terrain analysis using a digital map to select candidate observation sites and firing sites near this area of interest. These candidate positions are often near the edges of tree lines or buildings located on the map. In the robotic implementation, using a digital map to plan the mission is not always practical for small robots with limited processing power, so the human operator designates a set of geographic points to represent the area of interest and another set of geographic points to represent potential covered positions. The live robots (see section 1.2) plan their mission using these two sets of points.

After the initial planning phase of the algorithms, robotic team members assume their intended roles. In the second step of the behavior, the robots move to their initial positions. In our OTB OWF simulation, the observation site consists of two geographic points –the observation position and an associated concealed position that allows the observer to hide. The firing site consists of three geographic points—a firing position, a concealed position, and a preparatory position. The preparatory position allows the shooter to wait for instructions from the observer without being fully exposed. In the robotic implementation, the observation and firing sites are each

represented by geographic points indicating the approximate locations of yellow walls that can be used to conceal the robots. The robots navigate to these walls using Global Positioning System (GPS) sensors, color cameras, and sonar sensors. The GPS system guides the robots to within 3 m of the wall (the GPS system has a 3-m accuracy). Within this neighborhood, the robot navigates to the walls using its cameras. The robot stops when the sonar sensors indicate that the robot is close to the wall. The observer may need to maneuver around the wall to see the area of interest.

In the third step of the algorithm, the observer watches for targets. Once the target is identified (step four), the observer passes the information to the shooter (step five). Since safety is a concern for the robotic implementation, the observer can be required to pass the target information to the human operator for confirmation before passing the information to the shooter. In the OTB algorithm, target information is passed directly to the shooter.

After the shooter receives instructions to fire at the target, it moves to its firing position (step six, right-hand side of figure 1). Again, safety is a concern; the robot can be required to request human operator confirmation before it fires the gun in step seven. In the OTB implementation, the robots move to new positions in step six for the observer and step eight for the shooter. In the robotic implementation, the robots do not change positions.

1.2 Hardware Considerations

Two ATRV-series robots from iRobot were selected to demonstrate the behavior as surrogate robotic platforms for future tactical robotic systems. The robots are four-wheeled, skid-steered platforms that can be used indoors and outdoors. The ATRV and ATRV-Jr's sensors include visible spectrum cameras, ultrasonic range sensor array, GPS, an inertial measurement unit, a compass, and an inclinometer. A single line laser radar scanner provides obstacle location information to the make the navigation more robust. All sensor data analysis and mobility control is performed by a single onboard processor. The manufacturer has provided a software package, called Mobility, as an object-oriented control architecture for the robots. [2]

Figure 2 shows how we have simplified the demonstration environment to enable us to experiment with the behavior using laboratory robots with limited sensor capabilities and processing power. First, yellow walls, easily identified by the robots, simulate concealed locations that may be used as observation points and concealment for the shooter. Second, the target representing an enemy threat is a simple checkerboard pattern that the robots can identify easily. At the present time, the target does not move. Third, we operate the robots at slow speeds to allow time to process sensor information. We also simplified the planning process for the robots. They are provided a list of cover points and a list of watch points. These

[2] iRobot, Inc. *Mobility Robot Integration Software Users Guide*; iRobot, Inc: Jaffery, NH, 2000.

Figure 2. Behavior experiment using laboratory robots' simplifications allow us to focus on the algorithm and not be distracted by the integration of additional sensors, algorithms, or processors.

simplifications allow us to focus on the algorithm and not be distracted by the integration of additional sensors, algorithms, or processors.

The software required to demonstrate the OWF algorithm can be divided into three components—servers, an operator control unit (OCU), and the OWF application. Servers are independent software programs that communicate with sensors and actuators onboard the robot. Section 2 describes the servers required for the OWF behavior. The OCU, called the MONITOR program, is an independent process that allows operators to start, modify, and stop the OWF behavior. It is an optional element of the system—operators do not need to run the MONITOR program to demonstrate the OWF algorithms. The OWF application implements the algorithm described in the previous section. The MONITOR program is discussed in section 3.

2. Servers

Adopting a client/server viewpoint, the robot team can be considered as a collection of independent processes, called servers, which provide information and control the sensing and actuation systems on the robots. Application programs, such as the OWF program or human/robot interfaces, use these servers as command and communication interfaces to the underlying hardware. By using this approach, it is possible to have distributed applications communicating with the same hardware device simultaneously.

The Mobility software contains several servers used in this project. These include information servers for the sensors, such as the cameras, GPS, compass, sonar units, and the laser radar scanner. Command servers allow operators to control the robotic drive mechanism and the pan-tilt units for the cameras. We modified the pan-tilt servers to gain greater control over the motion of the unit. We also wrote a new camera server to interface with separately manufactured cameras connected with the ATRV robot.

We created two additional servers for this project – the information server (**InfoServer)** and the gun server (**GunServer)**. The **InfoServer** handles communications between the robotic and human team members of the OWF unit by maintaining a message board for the team. There are three types of messages—status messages, geographic information, and images. Each team member can post a short status message to report observations or to request help from another team member. Each status message consists of an 80-character string variable and timestamp that provides the time of message generation, in nanoseconds, and a message number. The type of geographic information posted includes the location of the region of interest, concealed sites, and mobility obstacles discovered as the robots drive through the region. Robots can post images to show the human operator suspected targets. The geographic information and the image information also include a timestamp and message number.

The **InfoServer** is an object derived from the ActiveSystemComponent class of the Mobility software library. Messages are placed on the server, or published, using the *new_sample* method of the InfoServer class. This makes the messages available to programs running on the robot or other computers within the local area network. Note that the server does not control message content—applications, such as the OWF behavior, determine the message set. The message set for this behavior is described in the ***GetMessage*** section. Messages are read by programs using the *update_sample* method of the InfoServer class. Both of these methods were inherited from the ActiveSystemComponent class and are used frequently to pass information to and from many servers included in the Mobility software package.

This server implements a broadcast strategy. A robot only publishes messages on its specific message line. Other robots must monitor that message line for new information. Note that this strategy does not guarantee message delivery.

The **GunServer** provides an interface to the weapon carried by the shooter. Currently, the "weapon" is a camera flash unit mounted in the center of the pan-tilt unit so that we can simplify safety considerations during the software development process. The weapon hardware communicates to the shooter's computer via a parallel port interface. The GunServer is a very simple interface that can accept two commands—*Fire* and *DoNotFire*. Like the **InfoServer,** the **GunServer** is an object derived from the ActiveSystemComponent class. Messages are published using the *new_sample* method for the GunServer class; messages are read by programs using *update_sample* method. In the near future, the camera flash unit will be replaced with a paint-ball marker. The GunServer will be modified to add safety features such as a power-down or disable.

Table 1 gives the list of servers required by our OWF software. The table provides a generic name for the server. Each robot may run a different copy of the server so the actual name must be unique. The table also provided the purpose of the server and the author.

Table 1. Servers required by the OWF software.

Server	Usual Host	Purpose	Author
ATRV	all robots: R3, R4, & R5	Control of mobility actuators	iRobot
laser	robot R3	Obstacle information	iRobot
sonar	all robots: R3, R4, & R5	Obstacle information	iRobot
framegrabber	robots: R4 & R5	Make Sony cameras images available	iRobot
GPS	all robots: R3, R4, & R5	GPS information	iRobot
Compass	all robots: R3, R4, & R5	Compass information	iRobot
Pan-Tilt	all robots R3, R4, & R5	Control of camera/gun pan-tilt unit	iRobot/ARL
VisionServer	robot R3	Make Panasonic camera available	ARL
InfoServer	OCU	Message Board for OWF unit	ARL
GunServer	robot R3	Control of camera flash unit or paint-ball gun	ARL

3. The MONITOR program

The MONITOR program serves as the OCU for the OWF behavior. It provides an efficient means to set parameters and pass information to the OWF behavior. Operators can start, modify, and stop the behavior from a graphical interface. MONITOR is also used as a diagnostic tool that allows the researcher to visualize information from several sensors at one time. Figure 3 shows the primary MONITOR window. The window consists of a button-bar at the top and a

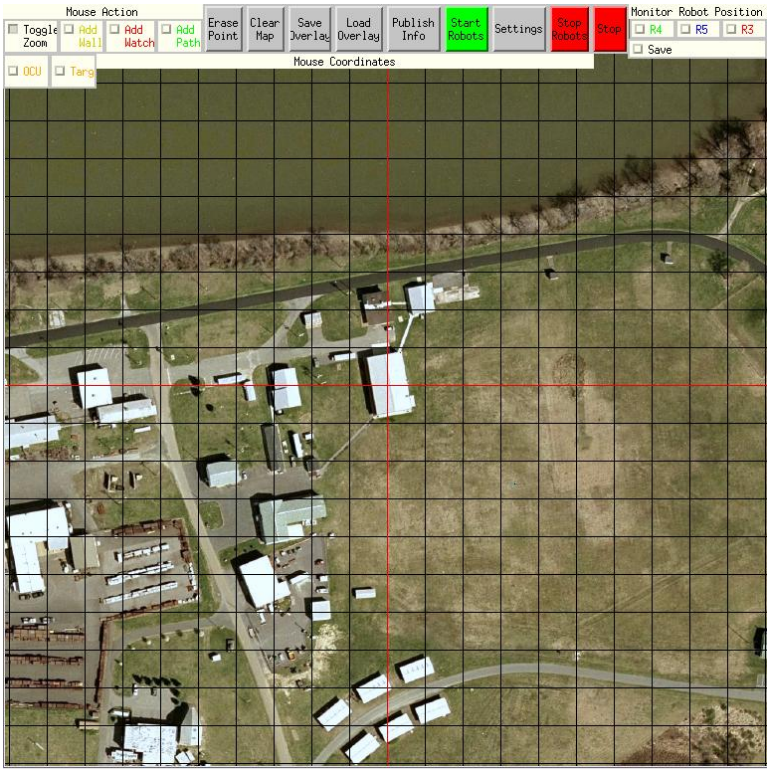

Figure 3. The main window of the MONITOR program.

map (or aerial photograph). The button-bar is subdivided into four areas. At the top left, a operator can select four possible mouse actions: "Toggle Zoom," "Add Wall," "Add Watch," and "Add Path." The Toggle Zoom selection allows the operator to switch between a regular and zoomed view of the map. The other three choices allow operators to add points to the map to indicate areas of cover (using the Add Wall selection), areas of interest (using the Add Watch selection), or GPS waypoints (using the Add Path selection). The set of wall, watch, and path points is considered as an information overlay for the map. The information is provided to the OWF behavior via the **InfoServer**. In the lower left of the button bar, the Mouse Coordinate section displays the geographic coordinates of the pointer. Coordinates are given in both universal transverse mercator and in latitude/longitude.

The gray, green, and red buttons in the center of the button-bar are control buttons. The first two buttons allow operators to change the location of points in the information overlay. The overlay can be saved to an external file named myOverlay for future reference; the "Load Overlay" button reads wall, watch, and/or path points from an external file (also named myOverlay). The "Publish Info" button sends geographic information to the **InfoServer**. The last four buttons start, modify, and stop the OWF behavior. The "settings" button allows operators to set options for the OWF program. Figure 4 shows the settings popup window that allows operators to assign roles to team members. This window allows operators to set up partial missions for

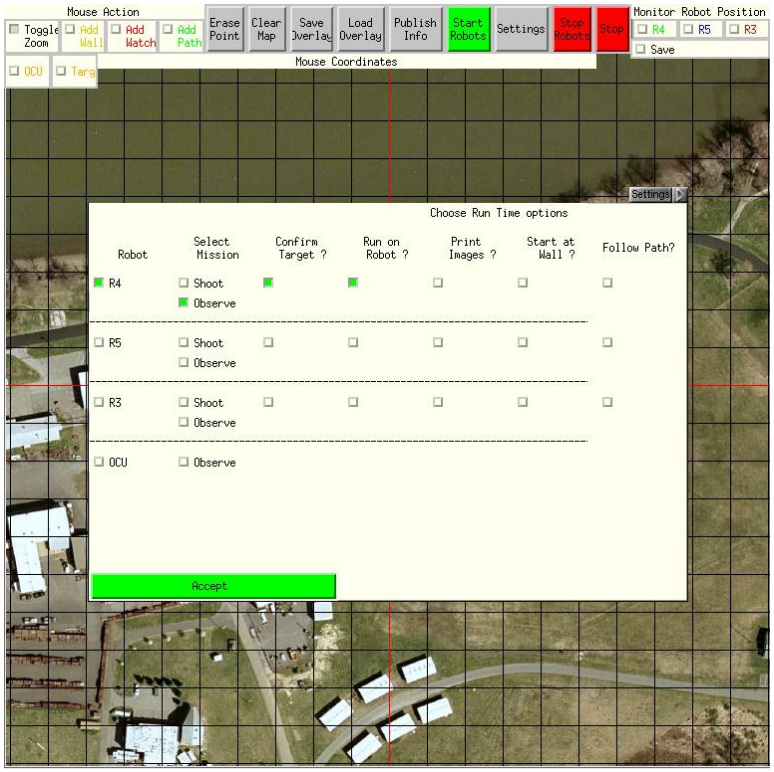

Figure 4. The settings popup window.

diagnostic purposes. The operate also uses this window to require the robots to get target confirmations before continuing the mission.

Back to figure 3, the "Start Robots" button starts the OWF mission as an independent process. The "Stop" button ends the MONITOR program and all the processes it started, whereas the "Stop Robots" button stops only the OWF program running on each of the robots.

The "Monitor Robot Position" section on the right-hand side of the button-bar allows the operator to display the position of the robotic vehicles on the map. The operator can also save this position data for later analysis.

As the OWF program executes, the MONITOR program uses the InfoServer to exchange messages with the robots. There are currently four message exchanges. The first exchange is initiated by the "Publish Info" button. The MONITOR program publishes the geographic points and a status message to notify the robots that the data is ready. The next two exchanges involve target confirmation; these exchanges only occur if the operator requires target confirmation. After a robot finds a target, it publishes both a status message and an image. Figure 5a shows the target popup window the MONITOR program displays. The "Confirm" and "Abort" buttons generate operator responses that are passed through the InfoServer to the OWF program. In the fourth type of message exchange, the robots pass information through the InfoServer to the monitor concerning the location of discovered (unmapped) mobility obstacles.

Figure 5. The target confirmation popup window.

4. Overwatching Fires Functions

This section describes the C++ functions developed for the OWF behavior. The functions are grouped by category—Image Processing, Sonar Processing, Movement, Communication, and Miscellaneous. Within each category, functions are described in alphabetical order. Each function description contains five sections—Function Prototype, Description, Input Variables, Output Variables, and Return Value. The Function Prototype provides the call syntax which gives the argument list and the return type. The arguments are described in the Input Variable and Output Variable section. The Description section provides a brief description of each function. Most of the functions return status information; defined constants with descriptive names are more useful as status information than the actual integer value of the constant. This section provides the defined constant names and their interpretation. The appendix provides a table of defined constants and their numerical values.

4.1 Image Processing

CenterTarget
Function Prototype
int *CenterTarget* (double InitialPanDegrees, double *FinalPanDegrees, int *ccx, int *ccy);
Description

CenterTarget uses the Pan-Tilt unit to turn the cameras until the suspected target is in the center of the image
Input Variables
InitialPanDegrees – the initial Pan position for the Pan-Tilt unit, in degrees.
Output Variables
FinalPanDegrees – the final position for the Pan-Tilt unit, in degrees.
(ccx, ccy) – position of the target center in the image given in pixels.
Possible Return Values
0 – meaningless.

ClassifyPixel

Function Prototype
int ClassifyPixel(int red, int green, int blue)
Description
ClassifyPixel determines the color of a pixel based on the RGB color scale. Possible return values are red, blue or neutral.
Input Variables
(red, green, blue) integer variables, with values in the range 0-255, describing the RGB color of the pixel
Possible Return Values
RED - the pixel is red
BLUE - the pixel is blue
NEUTRAL - the pixel is not red or blue.

FindNearestWall

Function Prototype
int FindNearestWall(int ImageNumber, int CameraNumber, int WallLocation,
int DesiredWallLocation,int *cx, int *cy, int *BinPixels int *NumberWallPixels)
Description
FindNearestWall finds the horizontal location of the nearest yellow wall in the image plane. The image plane is divided into thin rectangular cells (5 pixels wide 40 pixels high). Cells with more than 40 yellow pixels (using *IsPixelYellow* to classify the pixels) are considered part of a wall. *FindNearestWall* reports the location of the center of wall.
Input Variables
ImageNumber - integer variable giving the image number that is used to tag stored images.
CameraNumber- an integer variable specifying the desired camera.
WallPointLocation – last known x-position of the wall center in the image plane.
DesiredWallLocation - desired x-position of the wall center in the image plane.
Output Variables
cx,cy - location in the image plane of the point on the wall closest to the endpoint.
BinPixels – Highest number of yellow pixels within a single bin.
NumberWallPixels – total number of yellow pixels in the image.
Possible Return Values
NotEnoughPixels - there are not enough yellow pixels in the image to identify a wall.
EnoughPixels - there are enough yellow pixels in the image to identify a wall.

FindTargetInImage

Function Prototype
int *FindTargetInImage* (int ImageNumber, int CameraNumber, int DisplayPicture,

float period, int DesiredWayPointLocation, int *cx, int *cy, int *cbx, int *cby,
int *TotalHits, int *MaxBinHits)
Description
FindTargetInImage determines whether the target pattern is contained in the current image.
Input Variables
ImageNumber - integer variable giving the image number that is used to tag stored images.
CameraNumber- an integer variable specifying the desired camera.
DisplayPicture - an integer flag which determines if the system uses an external application to show images while the behavior is running.
period- floating point number that sets the image retrieval period in seconds.
DesiredWayPointLocation - desired x location of the endpoint in the image plane. This variable is passed to the graphics routine, *PrintImage* (described below), to draw a vertical reference line on the image.
Output Variables
cx, cy - location, in the image plane, of the endpoint using the entire image to estimate location.
cbx, cbx - location, in the image plane, of the endpoint using the image bin with the highest number of candidate points to estimate the location.
TotalHits - Total number of candidate points.
MaxBinHits - Largest number of candidate points within a single image bin.

Possible Return Values
EnoughPixels - indicates that there are enough candidate pixels to identify the endpoint in the image.
NotEnoughPixels- there are too few candidate pixels to identify the endpoint in the image.

GetImage

Function Prototype
int *GetImage*(int CameraNumber, double period)
Description
GetImage updates the stored image array from the camera specified by the CameraNumber variable. The period variable determines how often *new* images are retrieved. This allows calling routines flexibility in using images - the *GetImage* routine can be called from inside a high frequency loop, such as a driving loop, without requiring new images to be generated at the same frequency.
Input Variables
CameraNumber - integer variable specifying the desired camera.
period - double-length floating point number that sets the image retrieval period in seconds.
Possible Return Values
YES - A new image has been generated
No - no new image is available

IsPixelYellow

Function Prototype
int IsPixelYellow(int red, int green, int blue)
Description
IsPixelYellow determines the color of the pixel using the RGB color scale. The routine returns the integer constant YES if the pixel is yellow, NO otherwise.
Input Variables

(red, green, blue) integer variables describing the RGB color of the pixel.
Possible Return Values
YES - the pixel is yellow.
No - the pixel is not yellow.

max_of

Function Prototype
int *max_of* (int red, int green, int blue)
Description
The function *max_of* determines the maximum of the three integers red, green and blue. It is used by the function *RGBToHSI*.
Input Variables
(red,green,blue) – pixel color.
Output Variables
none
Possible Return Values
Maximum of the three input values represented as an integer.

min_of

Function Prototype
int *min_of* (int red, int green, int blue)
Description
The function *min_of* determines the minimum of the three integers red, green and blue. It is used by the function *RGBToHSI*.
Input Variables
(red,green,blue) – pixel color.
Output Variables
none
Possible Return Values
Minimum of the three input values represented as an integer.

RGB_HSI

Function Prototype
void RGB_HSI(int red, int green,int blue, float *hue, float *saturation, float *intensity)
;Description
The function *RGBToHSI* converts an image pixel from the Red-Green-Blue scale to the Hue-Saturation-Intensity scale
Input Variables
(red, green, blue) – pixel color using the Red-Green-Blue scale.
Output Variables
(hue, saturation, intensity) – pixel color using the Hue-Saturation-Intensity scale
Possible Return Values
None.

4.2 Sonar/Laser Processing

CheckSick
Function Prototype
int CheckSick(double *FrontDist,double *LeftDist,double *RightDist,double *RearDist,int *BestDirection)

Description
CheckSick grabs the most current set of laser line scanner readings to determine distance to nearby obstacles. The current set of distance readings, given in inches, are stored in an array called "Dist".
Output Variables
FrontDist – the address of a double-length floating point number giving the closest obstacle distance to the front of the vehicle, in inches.
RightDist - the address of a double-length floating point number giving the closest obstacle distance to the right side of the vehicle, in inches.
LeftDist - the address of a double-length floating point number giving the closest obstacle distance to the left side of the vehicle, in inches.
RearDist - the address of a double-length floating point number giving the closest obstacle distance to the rear of the vehicle, in inches.
BestDirection - the address of a double-length floating point number giving the best direction to move to avoid nearby obstacles.
Possible Return Values
SAFE - there are no objects within 15 inches of the robot
TooCloseFront - there is an object within 15 inches of the front of the robot.
TooCloseLeft there is an object within 15 inches of the left side of the robot.
TooCloseRight there is an object within 15 inches of the right side of the robot.
TooCloseRear there is an object within 15 inches of the rear of the robot.

CheckSonar

Function Prototype
int CheckSonar(double *FrontDist,double *LeftDist,double *RightDist,double *RearDist,int *BestDirection)
Description
CheckSonar grabs the most current set of sonar readings to determine distance to nearby obstacles. The current set of distance readings, given in inches, are stored in Dist array. *CheckSonar* divides the 17 sonars into 4 sets. For the ATRV-Jr™ they are: front (sonars 6, 7, 8, 9, and 10), rear (sonars 0 and 16), left (sonars 1, 2, 3, 4, and 5) and right(sonars 11, 12, 13, 14, and 15). For the ATRV™ they are: front (sonars 0, 1, 2, 4, 5, and 6), rear (sonars 9 and 10), left (sonars 11 and 3) and right(sonars 7 and 8). It returns the smallest distance for each of these sets.
Output Variables
FrontDist - the address of a double-length floating point number giving the closest obstacle distance to the front of the vehicle, in inches.
RightDist - the address of a double-length floating point number giving the closest obstacle distance to the right side of the vehicle, in inches.
LeftDist - the address of a double-length floating point number giving the closest obstacle distance to the left side of the vehicle, in inches.
RearDist - the address of a double-length floating point number giving the closest obstacle distance to the rear of the vehicle, in inches.
BestDirection - the address of a double-length floating point number giving the best direction to move to avoid nearby obstacles.
Possible Return Values
SAFE - there are no objects within 15 inches of the robot
TooCloseFront - there is an object within 15 inches of the front of the robot.

TooCloseLeft there is an object within 15 inches of the left side of the robot.
TooCloseRight there is an object within 15 inches of the right side of the robot.
TooCloseRear there is an object within 15 inches of the rear of the robot.

4.3 Movement

Bearing2Wpt
Function Prototype
int Bearing2Wpt(double EastCurrent, double NorthCurrent, double EastWpt,
double NorthWpt, double Threshold, double *BearingInDegrees)
Description
Bearing2Wpt gives the bearing from the vehicle to the current waypoint. *Bearing2Wpt* also calculates the distance to the waypoint to determine if the vehicle has reached the threshold for the current waypoint. If this condition has been met, the function returns FoundWayPoint which cues *FollowPath* to advance to the next way point in the route.
Input Variables
EastCurrent - double-length floating point variable specifying the current Easting of the vehicle.
NorthCurrent - double-length floating point variable specifying the current Northing of the vehicle.
EastWpt - double-length floating point variable specifying the Easting of the waypoint.
NorthWpt - double-length floating point variable specifying the Norting of the waypoint.
Threshold - double-length floating point variable specifying the distance threshold to the waypoint.
Output Variables
BearingInDegrees - double-length floating point variable specifying the bearing to the waypoint in degrees.
Possible Return Values
StillMoving - the distance threshold to the waypoint has not been reached.
FoundWaypoint – the distance threshold to the waypoint has been reached.

FacePoint
Function Prototype
int FacePoint(double EastCurrent, double NorthCurrent, double EastWpt,
double NorthWpt)
Description
FacePoint rotates the vehicle to face the point specified by (EastWpt, NorthWpt).
Input Variables
EastCurrent - double-length floating point variable specifying the current Easting of the vehicle.
NorthCurrent - double-length floating point variable specifying the current Northing of the vehicle.
EastWpt - double-length floating point variable specifying the Easting of the waypoint.
NorthWpt - double-length floating point variable specifying the Northing of the waypoint.
Threshold - double-length floating point variable specifying the distance threshold to the waypoint.
Output Variables
none
Possible Return Values
none

FollowPath

Function Prototype

int *FollowPath* (double PathNorth[25], double PathEast[25], double WallThreshold, double WptThreshold, double DefaultSpeed, int NumberPathPoints, int PrintFrequency)

Description

FollowPath Follows the path specified by the points { (PathEast[0], PathNorth[0]), (PathEast[1], PathNorth[1]), …, (PathEast[N], PathNorth[N]) } with N < 25 at a speed of DefaultSpeed while using the sonar information to avoid any obstacles. Obstacle avoidance is accomplished simply by turning away from the detected obstacle. No attempt is made to determine the shortest path length around the obstacle. The minimum front distance to a potential obstacle is calculated, and if the obstacle threshold distance is met, the relative angle to the obstacle is calculated. Based on this, the forward and angular velocities are then determined. The closer the obstacle, the harder the robot will turn to avoid impact. This algorithm uses 2 tolerances, WallThreshold and WptThreshold to determine how closely the robot follows the specified path. For the first N-1 points of the path, the robot moves towards its current point until the distance to the current path point is less than WptThreshold. The last point of the path, N, is treated as a special case – the distance between the robot and the Nth point is compared with the tolerance WallThreshold.

Input Variables

PathNorth– double-length floating point array of Northing coordinates

PathEast– double-length floating point array of Easting coordinates.

WallThreshold - double-length floating point variable the distance threshold to the wall.

WptThreshold - double-length floating point variable specifying the distance threshold to the waypoint.

DefaultSpeed - double-length floating point variable specifying the vehicle's default forward velocity.

NumberPathPoints - an integer that specifies the number of points in the PathNorth and PathEast arrays.

PrintFrequency – an integer that specifies the frequency of printed output.

Output Variables

none

Possible Return Values

ReachedDestination – value is returned when vehicle has reached its final waypoint position.

GetGPS

Function Prototype

int *GetGPS* (double *Lat, double *Long, double *northing, double *easting)

Description

GetGPS determines the location of the vehicle. It reads the latitude and longitude from the GPS sensor and calls *LatLonToUTM* to convert to the UTM scale used to calculate distances and display the robot's location on the map.

Input Variables

none

Output Variables

Lat – robot's Latitude coordinate provided by the GPS sensor.

Long – robot's Longitude coordinate provided by the GPS sensor.

Northing – robot's GPS northing coordinate in UTM
Easting – robot's GPS easting coordinate in UTM
Possible Return Values
none

GetHeading

Function Prototype
int *GetHeading* (double *HeadingDegrees, double *HeadingRad, double *dx, double *dy)
Description
GetHeading determines the robots orientation from the on-board compass. The compass measurement is based on magnetic north. *GetHeading* makes the offset correction to grid north, and returns this value in degrees and radians.
Input Variables
none
Output Variables
HeadingDegrees – robot's orientation in degrees
HeadingRad – robot's orientation in radians
dx – x coordinate of the unit vector describing the robots orientation , cos(HeadingRad)
dy – y coordinate of the unit vector describing the robots orientation , sin(HeadingRad)
Possible Return Values
OK

LatLonToUTM

Function Prototype
void *LatLonToUTM* (double lat, double lon, int *zone, double *northing, double *easting)
Description
LatLonToUTM converts the robot's latitudinal and longitudinal coordinates into UTM coordinates with its corresponding zone.
Input Variables
lat – robot's latitudinal coordinate
lon – robot's longitudinal coordinate
Output Variables
zone – robot's UTM zone
northing – robot's UTM northing coordinate
easting – robot's UTM easting coordinate
Possible Return Values
none

MoveBack

Function Prototype
void MoveBack()
Description
MoveBack backs the robot out of its firing position.
Input Variables
none
Possible Return Values
none

MoveIntoOpen

Function Prototype

void MoveIntoOpen()
Description
MoveIntoOpen moves the robot out of a position of cover to an observation position.
Input Variables
none
Possible Return Values
none

MoveToFiringPosition

Function Prototype
int *MoveToFiringPosition* (double FPNorth, double FPEast, double TargetNorth,
double TargetEast)
Description
MoveToFiringPosition navigates the robot out of its hiding position behind the yellow wall and into a position which allows a line of sight to a target position designated by the input variables TargetNorth and TargetEast.
Input Variables
FPNorth – a northing UTM coordinate suggested for the robot's firing position by the *PlanMission* function.
FPEast – an easting UTM coordinate suggested for the robot's firing position by the *PlanMission* function.
TargetNorth – a northing UTM coordinate
TargetEast – an easting UTM coordinate
Possible Return Values
ReachedDestination – the robot has reached its firing position.

MoveToWall

Function Prototype
int *MoveToWall* (int mission, double PathNorth[25], double PathEast[25],
int NumberPathPoints)
Description
MoveToWall follows the path defined by (PathNorth, PathEast) to a wall near the point (PathNorth[N], PatheEast[N]) where N is the last point in the path. The routine uses the GPS waypoint guidance in the function *FollowPath* until the robot is close enough to the wall, then switches to a visual guidance system that attempts to center the wall in the camera image. There are two different tolerances passed to *FollowPath*: FinalPointTolerance is the acceptable distance to the wall; IntermediatePointTolerance is the acceptable distance to other points on the path.
Input Variables
mission – SHOOTER or OBSERVER.
PathNorth[25] – an array of northing points that leads to a wall.
PathEast[25] – an array of easting points that leads to a wall.
NumberPathPoints – the number of path points that leads to a wall.
Possible Return Values
ProblemDetected - move cannot be completed.
ReachedDestination - move is successfully completed

4.4 Communication

GetMessage

Function Prototype
GetMessage(int robot)
Description
GetMessage gets a message from the server. It uses the variable **robot** to determine which message to retrieve.
Input Variables
robot - an integer variable designating the robot that published the message.
Possible Return Values
BEARING - message contains bearing information.
MOVING - message indicates the robot is in the MOVE state.
WATCHING -message indicates the robot is in the WATCH state.
STOPPED - the robot has terminated the mission.
DANGER - the robot has detected movement.
READY - Robot is ready to perform the mission.

PublishImage

Function Prototype
int *PublishImage*()
Description
PublishImage – sends an image to the InfoServer .
Input Variables
none
Possible Return Values
0 – meaningless.

PublishMessage

Function Prototype
int *PublishMessage* (int robot,char *msg)
Description
PublishMessage sends a message from the robot to the message server.
Input Variables
robot - an integer variable designating the robot publishing the message.
msg - a string variable containing the text of the message.
Possible Return Values
1 - meaningless.

PublishNewObstacles

Function Prototype
int *PublishNewObstacles*()
Description
PublishNewObstacles – sends a list of geographic coordinates for mobility obstacles to the InfoServer .
Input Variables
none
Possible Return Values
0 – meaningless.

4.5 Firing/Observing Functions

Fire
Function Prototype
int *Fire*(double North, double East);
Description
Fire activates the weapon.
Input Variables
(North, East) - target location
Possible Return Values
0 – meaningless.

Pan2Target
Function Prototype
int *Pan2Target* (double TargetNorth, double TargetEast, double *PanDegrees,
double *Range)
Description
Pan2Target finds the bearing from the vehicle's current position to the target in UTM coordinates. It returns the pan angle in degrees. For our purposes the tilt is fixed at zero. If the relative elevation between turret and target and the range to target are known, the inclinometer could provide enough additional information about the vehicle pose to calculate the tilt angle.
Input Variables
TargetNorth – target's UTM northing coordinate
TargetEast – target's UTM easting coordinate
Output Variables
PanDegrees – pan angle relative to the vehicle's heading from its current position to the target's UTM coordinates
range – the distance to the target
Possible Return Values
OK

ScanForTargets
Function Prototype
int *ScanForTargets*(int *cx, int *cy, double *PanDegrees, double *MapBearingDegrees);
Description
ProcessOptions sets options for the OWF behavior.
Input Variables
argv - a string array containing the command line arguments
argc - the number of command line arguments.
Possible Return Values
0 – meaningless.

SearchBearingLine
Function Prototype
int *SearchBearingLine* (double ObserverNorth, double ObserverEast,
double EndNorth, double EndEast,int *cx, int *cy,
double *PanDegrees,double *MapBearingDegrees
Description

SearchBearingLine is used by the Shooter to search for targets along a map bearing line supplied by the observer.
Input Variables
(ObserverNorth, ObserverEast) – the geographic location of the Observer
(EndNorth, EndEast) - the geographic location of the endpoint for the target bearing line.
Output Variables
int (cx,cy) – location of target in image pixel coordinates
PanDegrees – the current pan value, in degrees, for the pan and tilt unit
MapBearingDegrees – the estimate of target bearing from the firer.
Possible Return Values
0 – meaningless.

4.6 Miscellaneous

PickFiringPosition
Function Prototype
int *PickFiringPosition* (int Mission, int *NumberPathPoints);
Description
PickFiringPosition selects a firing position using the robot's current position and the location of the watch points.
Input Variables
none
Output Variables
(FPNorth, FPEast) – location of firing position.
Possible Return Values
0 – meaningless.
PlanMission
Function Prototype
int *PlanMission*(int Mission, int *NumberPathPoints);
Description
PlanMission produces a set of waypoints for the robot based on the robot's role and geographic overlay provided by the operator.
Input Variables
Mission - robot's role (Observer or Shooter)
Output Variables
NumberPathPoints - number of waypoints in the robot's plan.
Possible Return Values
0 – meaningless.
PointTurret
Function Prototype
int *PointTurret* (char *CoordSys,double Pan,double Tilt)
Description
PointTurret moves the pan and tilt unit in the relative coordinate system specified. Software limits are in place for rotations relative to the vehicle.
Input Variables
CoordSys – specifies either a rotation relative to the vehicle or relative to the turret.
(Pan, Tilt) – desired pan tilt location measured in degrees.

Possible Return Values
0 – meaningless.
PrintImage
Function Prototype
void PrintImage(int ImageNumber, int CameraNumber, int cx, int cy, int cbx, int cby,
int DesiredLocation)
Description
PrintImage writes an annotated image to a ascii portable pixmap file. In addition to the camera image, the saved image has a 20 x 20 grid, shown in white. Two pixels, one at (cx,cy) and the other at (cbx and cby) are highlighted in cyan and yellow, respectively. There is a vertical magenta line at DesiredLocation for reference. Images are tagged with the robot number, Camera number and an Image number so that they can be easily organized for post-processing.
Input Variables
ImageNumber - integer variable giving the image number that is used to tag stored images
CameraNumber- an integer variable specifying the desired camera.
cx,cy - image pixel to be highlighted in cyan.
cbx,cby - image pixel to be highlighted in yellow.
DesiredLocation -Location in the image plane of a aertical reference line to be drawn in magenta.
Possible Return Values
None
PrintYellowImage
Function Prototype
void PrintYellowImage(int ImageNumber, int CameraNumber,
int WallLocation, int DesiredWallLocation ,int LoX, int HiX)
Description
PrintYellowImage writes the current processed image for the CameraNumber to a ascii portable pixmap file. The image shows yellow wall pixels, neutral pixels and a grid. The image also shows vertical reference lines at WallLocation and DesiredWallLocation. Images are tagged with the robot number, Camera number and an Image number so that they can be easily organized for post-processing.
Input Variables
ImageNumber - integer variable giving the image number that is used to tag stored images
CameraNumber- an integer variable specifying the desired camera.
WallLocation - integer variable giving location of the vertical wall edge closest to the endpoint.
DesiredWalLocation - integer variable giving desired location of the wall in the image plane.
LoX,HiX - integer variables specifying the boundaries of the search region in the image plane. Typically, one boundary is set to the current location of the endpoint and the other boundary is set to the appropriate edge of the image plane
Possible Return Values
None
ProcessOptions
Function Prototype
int *ProcessOptions*(int argc, char *argv[])
Description
ProcessOptions sets options for the OWF behavior.
Input Variables

argv - a string array containing the command line arguments
argc - the number of command line arguments.
Possible Return Values
None
ShakeHead
Function Prototype
int *ShakeHead*(int argc, char *argv[])
Description
ShakeHead – moves the Pan-Tilt unit up and down as a debugging feature to allow the operator to know that a command has been received
Input Variables
argv - a string array containing the command line arguments.
argc - the number of command line arguments.
Possible Return Values
None
StartServers
Function Prototype
int *StartServers*(int argc, char *argv[])
Description
StartServers links local variables to the servers necessary to run the behavior. There are seven servers used on the robot. The DriveCommand server sends commands to the driving system. The Odometery server provides position information. The Sonar server provides data from the sonar array. Two Camera servers provide images from the cameras. The Pan-Tilt server allows control of the camera gaze. The Compass server provides compass information. The two remaining servers, the Information server and the Map server are hosted by other computer systems on the local area network used by the robots. The Information servers allows messages to be passed between the robots. The Map server maintains a shared obstacle map used for debugging purposes.
Input Variables
argv - a string array containing the command line arguments
argc - the number of command line arguments.
Possible Return Values
None

---

5. Conclusions

This report has presented a guide to the software developed for the OWF behavior implemented on iRobots' ATRV/ATRV-Jr platforms. It presents a short description of the behavior algorithm and a detailed description of the servers and functions used to implement the algorithm.

In our future work, we are interested in using the OWF behavior as an experimental system. We are interested in studying the amount of time that the OWF team requires to acquire and fire at targets. This timeline depends on communication delays in the system, robot processor speed,

and on the level of involvement for the human operator. Our experimental area is too small to actually affect communications but we can delay messages to simulate communication delays. As we make the image processing algorithms more sophisticated, we expect to impact processing time. Right now, there is only one target in the experimental setup. The human's role is to confirm that the robots have identified this target. By introducing multiple targets and false targets, the operator's response time may change. Allowing the targets to move could also effect the overall mission timeline.

In our future work, we will also incorporate more realistic sensor algorithms. In particular, we plan to use more realistic hiding locations in the future. We will modify the *FindNearestWall* routine so that it uses vertical edges, shape, and color information to identify possible hiding locations.

Right now the robots do very little planning to determine their next course of action. By incorporating a world map, from the InfoServer, the robots could plan their moves more effectively. We will also address this issue in our future research.

Appendix. Named Constants

The following is a list of the defined constants used in the Overwatching Fires software.

Constant Name	Value	Meaning
ABORT	1017	Message Content: OCU aborts fire mission.
BLUE	201	Pixel color is blue.
Blocked	-500	Planning constant: map location is blocked.
BOTH	7202	Confirm targets for observers and shooters.
CanNotSee	0	Planning constant: map location cannot be seen from robot location.
CanSee	1	Planning constant: map location can be seen from robot location.
Clear	-501	Planning constant: map location is clear.
CompletedMove	501	Move has successfully completed.
CONTINUE	1018	Message Content:moving.
DANGER	1005	Message Content: Team member encounters enemy units.
FoundWayPoint	105	Robot is close enough to designated waypoint.
EnoughPixels	104	Image does not contain enough target/wall pixels for analysis.
Halted	109	Robot has halted.
MAP_DATA_AVAILABLE	1012	Message Content:Ocu has published overlay points.
MOVING	1002	Message Content:moving.
NEGATIVE	-1	Endpoint is on the right of the robot.
NEUTRAL	203	Pixel color is not a target color.
NO	0	Unsuccessful completion of function.
NormalForwardSpeed	0.6	Normal driving speed in m/s.
NONE	7203	Do not confirm targets.
NotEnoughPixels	103	Image does not contain enough target/wall pixels for analysis.
OBSERVER	7200	Team member role: observer.
ObstacleDetected	13	There is an obstruction near the robot.
OCU	1	Team member role: human interface unit.
OK	101	Function has successfully completed.
POSITION	1015	Message Content:robot is sending position data.
POSITIVE	1	Endpoint is on the left of the robot.
ProblemDetected	102	Function encounters a problem and cannot complete successfully.
ReachedDestination	106	Reached destination such as wall or GPS waypoint.
READY	1006	Message Content:Robot ready for the mission to begin.

RED	200	Pixel color is red.
RobotR3	3	Robot R3.
RobotR4	4	Robot R4.
RobotR5	5	Robot R5.
SAFE	300	The robot can safely move.
SafeDistance	15.0	Maximum safe distance from obstacles in inches.
SHOOT	1010	Message Content:Orders to fire at target.
SHOOTING	1011	Message Content:Robot is shooting target.
SHOOTER	7201	Team member role: shooter.
StartingMove	107	Robot is beginning movement.
StillMoving	108	Robot is still traveling.
STOPPED	1003	Message Content:stopped.
SomethingInTheWay	502	Move cannot be completed.
TARGET	1014	Message Content:Robot see a potential target.
TARGET_CONFIRMED	1013	Message Content:OCU confirms target.
TooCloseFront	301	Obstacle near the front of the robot.
TooCloseLeft	303	Obstacle near the left of the robot.
TooCloseRear	302	Obstacle near the rear of the robot.
TooCloseRight	304	Obstacle near the right of the robot.
UNKNOWN	2000	Unknown message content.
WAITING	1016	Message Content:robot waiting for orders.
WallPoint	-100	Planning constant: map location is a known wall point.
WATCHING	1004	Message Content:robot watching for targets.
WatchPoint	-200	Planning constant: map location is a known area of interest.
YELLOW	202	Pixel color is yellow.
YES	1	Successful completion of function.

NO. OF COPIES ORGANIZATION

1 (PDF ONLY) DEFENSE TECHNICAL INFORMATION CTR
DTIC OCA
8725 JOHN J KINGMAN RD
STE 0944
FORT BELVOIR VA 22060-6218

1 US ARMY RSRCH DEV & ENGRG CMD
SYSTEMS OF SYSTEMS INTEGRATION
AMSRD SS T
6000 6TH ST STE 100
FORT BELVOIR VA 22060-5608

1 INST FOR ADVNCD TCHNLGY
THE UNIV OF TEXAS
AT AUSTIN
3925 W BRAKER LN STE 400
AUSTIN TX 78759-5316

1 US MILITARY ACADEMY
MATH SCI CTR EXCELLENCE
MADN MATH
THAYER HALL
WEST POINT NY 10996-1786

1 DIRECTOR
US ARMY RESEARCH LAB
IMNE ALC IMS
2800 POWDER MILL RD
ADELPHI MD 20783-1197

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRD ARL CI OK TL
2800 POWDER MILL RD
ADELPHI MD 20783-1197

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRD ARL CS IS T
2800 POWDER MILL RD
ADELPHI MD 20783-1197

NO. OF COPIES ORGANIZATION

ABERDEEN PROVING GROUND

1 DIR USARL
AMSRD ARL CI OK TP (BLDG 4600)

NO. OF COPIES	ORGANIZATION
1	DOD JOINT CHIEF OF STAFF DEFENSE TECH INFO CTR J39 CAPABILITIES DIV J M BROWNELL THE PENTAGON RM 2C865 WASHINGTON DC 20301
1	DIR CIA D MOORE WASHINGTON DC 20505-0001
3	PM ABRAMS TANK SYS SFAE GCS AB COL KOTCHMAN P LEITHEISER H PETERSON WARREN MI 48397-5000
1	PM M1A2 SFAE GCS AB R LOVETT WARREN MI 48397-5000
1	PM M1A1 SFAE GCS AB L C MILLER JR WARREN MI 48397-5000
1	PM BFVS SFAE GCS BV C L MCCOY WARREN MI 48397-5000
1	PM BFVS ATZB BV C BETEK FORT BENNING GA 31905
1	PM M2/M3 BFVS SFAE GCS BV J MCGUINESS WARREN MI 48397-5000
3	PM BCT SFAE GCS BCT R D OGG JR J GERLACH T DEAN WARREN MI 48397-5000

NO. OF COPIES	ORGANIZATION
1	PM IAV SFAE GCS BCT J PARKER WARREN MI 48397-5000
1	PM NIGHT VISION/RSTA SFAE IEW&S NV COL BOWMAN FORT BELVOIR VA 22060-5806
1	NIGHT VISION & ELECTRONIC SENSORS DIR A F MILTON 10221 BURBECK RD SUITE 430 FORT BELVOIR VA 22060-5806
1	COMMANDER US ARMY TRADOC ATINZA R REUSS BLDG 133 FORT MONROE VA 23651
1	OFC OF THE SECY OF DEFENSE CTR FOR COUNTERMEASURES M A SCHUCK WHITE SANDS NM 88002-5519
1	US SOCOM SOIO JA F J GOODE 7701 TAMPA POINT BLVD BLDG 501 MCDILL AFB FL 33621-5323
1	COMMANDER US ARMY ARMOR CTR & FT KNOX TSM/ABRAMS D SZYDLOSKI FORT KNOX KY 40121
1	COMMANDER US AMBL J HUGHES FORT KNOX KY 40121

NO. OF COPIES ORGANIZATION

1 DIR OF COMBAT DEV
ATZK FD
W MEINSHAUSEN
BLDG 1002 RM 326
1ST CALVARY DIV RD
FORT KNOX KY 40121-9142

1 COMMANDING OFFICER
MARINE CORPS INTEL ACTIVITY
W BARTH
3300 RUSSELL RD SUITE 250
QUANTICO VA 22134-5011

2 COMMANDER
US TACOM ARDEC
AMSTA AR TD
M DEVINE
M FISETTE
PICATINNY ARSENAL NJ
07806-5000

4 COMMANDER
US TACOM ARDEC
AMSTA AR FSA S
S R KOPMANN
H KERWIEN
K JONES
A FRANCHINO
PICATINNY ARSENAL NJ
07806-5000

1 COMMANDER
AMSTA AR FSA P
D PASCUA
PICATINNY ARSENAL NJ
07806-5000

1 COMMANDER
AMSTA AR FSA M
J FENECK
PICATINNY ARSENAL NJ
07806-5000

2 COMMANDER
AMSTA AR FSP
D LADD
M CILLI
PICATINNY ARSENAL NJ
07806-5000

NO. OF COPIES ORGANIZATION

6 COMMANDER
AMSTA AR CCH A
M PALTHINGAL
A VELLA
E LOGSDON
R CARR
M MICOLICH
M YOUNG
PICATINNY ARSENAL NJ
07806-5000

1 COMMANDER
AMSTA AR QAC
R SCHUBERT
PICATINNY ARSENAL NJ
07806-5000

2 COMMANDER
AMSTA AR FSP G
A PEZZANO
R SHORR
PICATINNY ARSENAL NJ
07806-5000

1 COMMANDER
AMSTA AR FSA T
A LAGASCA
PICATINNY ARSENAL NJ
07806-5000

1 COMMANDER
AMSTA AR FSP I
R COLLETT
PICATINNY ARSENAL NJ
07806-5000

2 COMMANDER
AMSTA AR WE C
R FONG
S TANG
PICATINNY ARSENAL NJ
07806-5000

1 COMPUTER SCI AND ENGR
UNIV OF SOUTH FLORIDA
R MURPHY
4202 E FOWLER AVE ENB342
TAMPA FL 33620-5399

1 APPLD PHYSICS LAB
11100 JOHNS HOPKINS RD
T NEIGHOFF
LAUREL MD 20723-6099

NO. OF COPIES	ORGANIZATION
1	SAIC KA JAMISON PO BOX 4216 FT WALTON BEACH FL 32549
1	PEO GCS SFAE GCS C GAGNON WARREN MI 48397-5000
1	PEO GCS SFAE GCS W A PUZZUOLI WARREN MI 48397-5000
1	PEO GCS SFAE GCS BV J PHILLIPS WARREN MI 48397-5000
1	PEO GCS SFAE GCS LAV T LYTLE WARREN MI 48397-5000
1	PEO GCS SFAE GCS AB SW DR PATTISON WARREN MI 48397-5000
1	PEO GCS SFAE GCS AB LF LTC PAULSON WARREN MI 48397-5000
1	PEO GCS SFAE GCS LAV M T KLER WARREN MI 48397-5000
1	PEO GCS SFAE GCS LAV FCS ASOKLIS WARREN MI 48397-5000
2	COMMANDER US ARMY TACOM AMSTA TR R MCCLELLANC BAGWELL WARREN MI 48397-5000
11	COMMANDER US ARMY TACOM AMSTA TR R J PARKS S SCHEHR D THOMAS C ACIR J SOLTESZ S CAITO K LIM J REVELLO B BEAUDOIN B RATHGEB M CHAIT WARREN MI 48397-5000
8	COMMANDER AMSTA CM XSF R DRITLEIN HENDERSON HUTCHINSON SCHWARZ S PATHAK R HALLE J ARKAS G SIMON WARREN MI 48397-5000
3	PEO PM MORTAR SYS SFAE AMO CAS IFM L BICKLEY M SERBAN K SLIVOVSKY PICATINNY ARSENAL NJ 07860-5000
1	PEO PM MORTAR SYS SFAE GCS TMA R KOWALSKI PICATINNY ARSENAL NJ 07860-5000
1	PEO PM MORTAR SYS SFAE GCS TMA PA E KOPACZ PICATINNY ARSENAL NJ 07860-5000
3	MIT LINCOLN LAB J HERD G TITI D ENGREN 244 WOOD ST LEXINGTON MA 02420-9108

NO. OF COPIES ORGANIZATION

2 THE UNIV OF TEXAS
AT AUSTIN
I MCNAB
S BLESS
PO BOX 20797
AUSTIN TX 78720-2797

1 UNIV OG NEBRASKA
S FARRITOR
N118 WALTER SCOTT ENGR CTR
LINCOLN NE 68588-0656

1 INNOVATIVE SURVIVABILITY TECH
J STEEN
PO BOX 1989
GOLETA CA 93116

1 SUNY BUFFALO
ELECTRL ENGR DEPT
J SARJEANT
PO BOX 601900
BUFFALO NY 14260-1900

1 GENERAL DYNAMICS LAND SYS
D GERSDORFF
PO BOX 2074
WARREN MI 49090-2074

1 COMMANDER
US ARMY CECOM
W DEVILBISS
BLDG 600
FORT MONMOUTH NJ 07703-5206

1 MARCORSYSCOM CBG
J DOUGLAS
QUANTICO VA 22134-5010

2 COMMANDER
USAIC
ATZB CDF
J LANE
D HANCOCK
FORT BENNING GA 31905

1 DIRECTOR
US ARMY RSCH LAB
AMSRL SL EA
R CUNDIFF
WSMR NM 88001-5513

NO. OF COPIES ORGANIZATION

1 DIRECTOR
US ARMY RSCH LAB
AMSRL SL EM
J THOMPSON
WSMR NM 88001-5513

4 UNITED DEFNS ADV DEV CTR
K GROVES
J FAUL
T WINANT
V HORVATICH
328 BROKAW RD
SANTA CLARA CA 95050

2 NORTHROP GRUMMAN CORP
A SHREKENHAMER
D EWART
1100 W HOLLYVALE ST
AAUSA CA 91702

1 COMMANDER
US ARMY AMCOM
AMSAM RD ST WF
D LOVELACE
REDSTONE ARSENAL AL
35898-5247

1 OFC OF THE SECY OF DEFNS
ODDRE R&T
G SINGLEY
THE PENTAGON
WASHINGTON DC 20301-3080

1 US MILITARY ACADEMY
MATHEMATICAL SCIENCES
CTR OF EXCELLENCE
MDN A
MAJ HUBER
THAYER HALL
WEST POINT NY 10996-1786

2 DIRECTOR
US ARMY WATERWAYS
EXPER STATION
R AHLVIN
3909 HALLS FERRY RD
VICKSBURG MS 39180-6199

2 NATL INST STAN AND TECH
K MURPHY
100 BUREAU DR
GAITHERSBURG MD 20899

NO. OF COPIES ORGANIZATION

3 COMMANDER
US ARMY MMBL
J BURNS
BLDG 2021
BLACKHORSE REGIMENT DR
FORT KNOX KY 40121

2 DIRECTOR
NASA JET PROPULSION LAB
L MATHIES
K OWENS
4800 OAK GROVE DR
PASADENA CA 91109

1 DIRECTOR
AMCOM MRDEC
AMSMI RD W
C MCCORKLE
REDSTONE ARSENAL AL 35898-5240

1 COMMANDER
US ARMY INFO SYS ENGRG CMD
ASQB OTD
F JENIA
FT HUACHUCA AZ 85613-5300

1 COMMANDER
US ARMY NATICK RDEC
ACTING TECHNICAL DIR
SSCNC T
P BRANDLER
NATICK MA 01760-5002

1 COMMANDER
ARMY RSCH OFC
4300 S MIAMI BLVD
RSCH TRIANGLE PARK
NC 27709

1 COMMANDER
US ARMY TRADOC
BATTLE LABE INTEGRATION
7 TECH DIR
ATCD BJ
A KLEVECZ
FT MONROE VA 23651-5850

1 DARPA
D KASPAR
3701 N FAIRFAX DR
ARLINGTON VA 22203-1714

NO. OF COPIES ORGANIZATION

ABERDEEN PROVING GROUND

1 PM ODS
SFAE CBD
B WELCH
BLDG 4475
APG MD 21010-5424

1 COMMANDER
US ARMY ATC
VIRTUAL PROVING GROUND TEAM
J CORDE
400 COLLERAN RD BLDG 321
APG MD 21005-5000

1 COMMANDER
USAATC
STEAC CO
ELLIS
APG MD 21005-5001

1 COMMANDER
USAATC
STEAC TD
J FASIG
APG MD 21005-5001

1 COMMANDER
USAATC
STEAC TE
H CUNNINGHAM
APG MD 21005-5001

1 COMMANDER
USAATC
STEAC RM
A MOORE
APG MD 21005-5001

3 DIRECTOR
USAMSAA
AMSRD AMS D
M MCCARTHY
B SIEGEL
P TOPPER
APG MD 21005-5067

1 COMMANDER
USAATC
STEAC
ELLIS
APG MD 21005-5000

NO. OF COPIES	ORGANIZATION
1	COMMANDER USAATC STEAC TD J FASIG APG MD 21005-5000
1	COMMANDER USAATC STEAC TE H CUNNINGHAM APG MD 21005-5000
1	COMMANDER USAATC STEAC RM C A MOORE APG MD 21005-5000
1	COMMANDER USAATC STEAC TE F P OXENBERG APG MD 21005-5000
30	DIR USARL AMSRD ARL WM J BORNSTEIN B BURNS B RINGER T ROSENBERGER E SCHMIDT J SMITH C SHOEMAKER AMSRD ARL WM B W CIEPIELLA A HORST AMSRD ARL WM BA D LYONS AMSRD ARL WM BC P PLOSTINS AMSRD ARL WM BD B FORCH AMSRD ARL WM BF M BARANOSKI H EDGE M FIELDS G HAAS T HAUG W OBERLE R PEARSON D WILKERSON AMSRD ARL WM TC R COATES

NO. OF COPIES	ORGANIZATION
	AMSRD ARL WM TE P BERNING C HUMMER T KOTTKE M MCNEIR A NIILER J POWELL G THOMSON AMSRD SL BG M ENDERLEIN AMSRD SL EM C GARRETT